САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Кондратьева Александра Васильевна

Дипломная работа

Построение оптимального маршрута и его визуализация с помощью WebGL

Научный руководитель:
ст. преп. Уланов А.В.

Санкт-Петербург

2016

SAINT-PETERSBURG STATE UNIVERSITY

DEPARTMENT OF TECHNOLOGY OF PROGRAMMING

Aleksandra Kondrateva

Graduation Thesis

The Traveling Salesman Problem and its visualization with WebGL

Scientific supervisor:
Senior Lecturer
Aleksandr Ulanov

Saint Petersburg

2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача построения оптимального маршрута, впервые поднятая в 1832 году в книге «Коммивояжер – как он должен вести себя и что должен делать для того, чтобы доставлять товар и иметь успех в своих делах – советы старого курьера», является актуальной и на сегодняшний день – разрабатываются новые методы решения задачи, реализуются программы, которые позволяют работать с количеством узлов близким к миллиону за приемлемое время. Неугасаемый интерес к этой задачи обусловлен разнообразием применений ее на практике. Поиск оптимального пути широко используется во всех задачах транспортной логистики, на производстве – в виде задач минимизации времени переналадки и при работе дыропробивных прессов. [1]

Вопросы оптимизации маршрута поднимаются в трудах таких известных деятелей математики как Уильям Роуэн Гамильтон, Джордж Данциг, Ричард Мэннинг Карп, Дэвид Аплгейт, Герхард Райнельт. [1]

Несмотря на обширную теоретическую базу, к сожалению, представлено не так много приложений, позволяющих людям, далеким от математики и программирования, использовать существующую разработки для решения практических задач. Пользователь, который хочет обойти n-ное количество мест за минимальное время, используя такие популярные картографические сервисы как «Яндекс.Карты» [2] и «Google Карты» [3], не может решить поставленную задачу, так как маршрут в приложении строится по заранее заданному в определенном порядке списку мест. Цель данной работы заключается в интегрировании алгоритма задачи коммивояжера в Google.Maps с использованием технологии WebGL [4] на примере приложения поиска оптимального маршрута между достопримечательностями Санкт-Петербурга.

Для достижения указанной цели поставлены следующие задачи:

1. рассмотреть основные алгоритмы поиска оптимального маршрута;
2. провести анализ данных алгоритмов и выбрать один из них для практической реализации;
3. написать код выбранного алгоритма, используя JavaScript;
4. на основе полученных данных анализа разработать приложение, позволяющее пользователям отметить на карте достопримечательности и получить оптимальный путь, наглядно показанный на Google.Maps.

Актуальность данной работы заключается в поставленной цели – анализа алгоритмов, оптимальных для поставленной задачи, и создании удобного приложения по поиску оптимального пути для гидов, экскурсоводов и нуждающихся в быстром доступе к разнообразным маршрутам туристов. Практическая значимость работы выражается в потенциале для дальнейшего развития и коммерциализации приложения.

Дипломная работа разделена на три главы. В первой главе «Обзор задачи» представлен обзор предметной области, описывается функциональность приложения, определяются границы проекта. Во второй главе «Математическая основа проекта» проводится анализ алгоритмов для решения задачи. В третьей главе «Практическая реализация и результаты работы» рассказано, какие инструменты использовались при создании проекта, анализируются полученные данные.

ГЛАВА 1. ОБЗОР ЗАДАЧИ

Первая глава состоит из обзора предметной области, включающего в себя описание и историю задачи коммивояжера, а также анализа уже существующих приложений. Ставится задача диплома, формулируются требования к ее реализации.

1.1. Задача коммивояжера

Комбинаторика – это раздел математической науки, изучающей комбинации и их количество, составленные по тем или иным условиям из заданных объектов. Впервые вопросы, которые изучает комбинаторика, были подняты в трудах математиков Древней Греции, Индии и Китая. Интерес к предмету увеличился в 19-ом и 20-ом веках в связи с развитием теории графов. У комбинаторики есть много применений в других областях математики, включая теорию графов, программирование, криптографию и теорию вероятности. Среди ведущих математиков этой области можно выделить Блеза Паскаля (1623-1662), Якоба Бернулли (1654-1705), Леонхарда Эйлера (1707-1783) и Пала Эрдёша (1913-1996). [5]

Задача коммивояжера (Travelling salesman problem, TSP) – классическая задача комбинаторики. Чтобы нагляднее показать ее значимость, ниже приведена краткая история решения поставленной задачи.

Математические проблемы, связанные с задачей коммивояжера, были изучены в 1800-ых годах ирландским математиком сэром Уильямом Роуэном Гамильтоном и британским математиком Томасом Пенингтоном Киркменом. В 1857 Гамильтон создал игру Икосиан, в правилах которой сказано, что участники должны соединить 20 точек додекаэдра так, чтобы каждая точка использовалась не более одного раза, также конечная точка пути должна совпадать с исходной. Эта игра легла в основу появления гамильтонова графа. [6]

Задача поиска оптимально пути была впервые рассмотрена с математической точки зрения в 1930-ых годах математиком и экономистом Карлом Менджером в Вене. В 1940-ых статистики Мэхаланобис, Джессен, Гош и Маркс пытались найти применение задаче коммивояжера в сельскохозяйственном секторе, что привело к ее популяризации – начиная с середины 1950ых годах методы решения задачи стали публиковаться в научных журналах.

Хотя задача коммивояжера проста для понимания, ее крайне сложно решить. [7] В 1972 Ричард М. Карп показал, что задача гамильтонова цикла принадлежит к классу NP-полных задач (Nondeterministic Polynomial time), которые подразумевает NP-сложность задачи TSP. [8] Это открытие дало научное объяснение очевидной вычислительной трудности нахождения оптимальных маршрутов.

Методы решения TSP становились более сложными, количество городов возрастало. Ниже представлена краткая история этапов решения задачи коммивояжера.

В 1954 Данциг, Фалкерсон и Джонсон издали описание метода для решения TSP и практически проиллюстрировали его, решив задачу с 49 городами, соединив таким образом города каждого штата Америки. В 1971 году Хельд и Карп решили задачу поиска оптимального пути между 64 городами. Позже в 1977 году, Мартин Гротчел построил путь между 120 немецкими городами. В 1973 Лин и Керниган нашли применение задачи TSP в инженерии – они соединили 318 пунктов, полученных в результате резки лазером. [9]

В 1987 году Голландом и Гроешел была решена задача с 666 городами, позже в этом же году Падберг и Ринальди нашли оптимальный тур, связывающий уже 2392 города. В 1994 году количество городов увеличилось до 7397, спустя 10 лет количество городов достигло до 24978. В 2006 было получено решение задачи коммивояжера с 85900 городами. [9]

1.2. Анализ существующих сервисов и их недостатки

Популярные картографические сервисы вроде Яндекс.Карты и Google Maps не предоставляют пользователям возможность поиска оптимально пути. При вводе нескольких координат сервис выстраивает маршрут в том порядке, в котором данные были введены. Пользователи могут выбирать средства передвижения (на машине, пешком, на наземном транспорте), но все эти настройки влияют исключительно на варианты построения маршрута между его фиксированными точками.

Анализ, проведенный путем сравнением десятков как русскоязычных, так и зарубежных картографических сервисов показывает, что среди самых популярных вариантов только у одного доступна функция построения оптимально пути, и далеко не всегда она работает корректно. Ниже представлен подробный отчет о трех наиболее достойных внимания сервисах и их недостатках.

Среди русскоязычных сервисов в первую очередь можно выделить Meganavigator [10]. Он включает в себя конструктор карт, визуализацию GPS треков и построение оптимальных маршрутов. Сервис представляет из себя сайт со встроенной Яндекс.Картой и полями для ввода пунктов маршрута (Рис 1.1). Тестирование функционала показало наличие существенных недостатков:

1. несмотря на то, что Meganavigator позиционирует себя как сервис для построения автомобильных, велосипедных и пешеходных маршрутов, на сайте отсутствует выбор средства передвижения; режим, выставленный по умолчанию, соответствует передвижению на автотранспортном средстве;
2. у сервиса не предусмотрены подсказки при вводе адреса, что затрудняет работу с ним;

3. при построении маршрута первый пункт выбирается произвольно из заданного списка адресов; нет возможности задавать адрес как первый по умолчанию;
4. при выборе пунктов на интерактивной карте без ввода адресов в текстовые поля, маршрут не строиться.

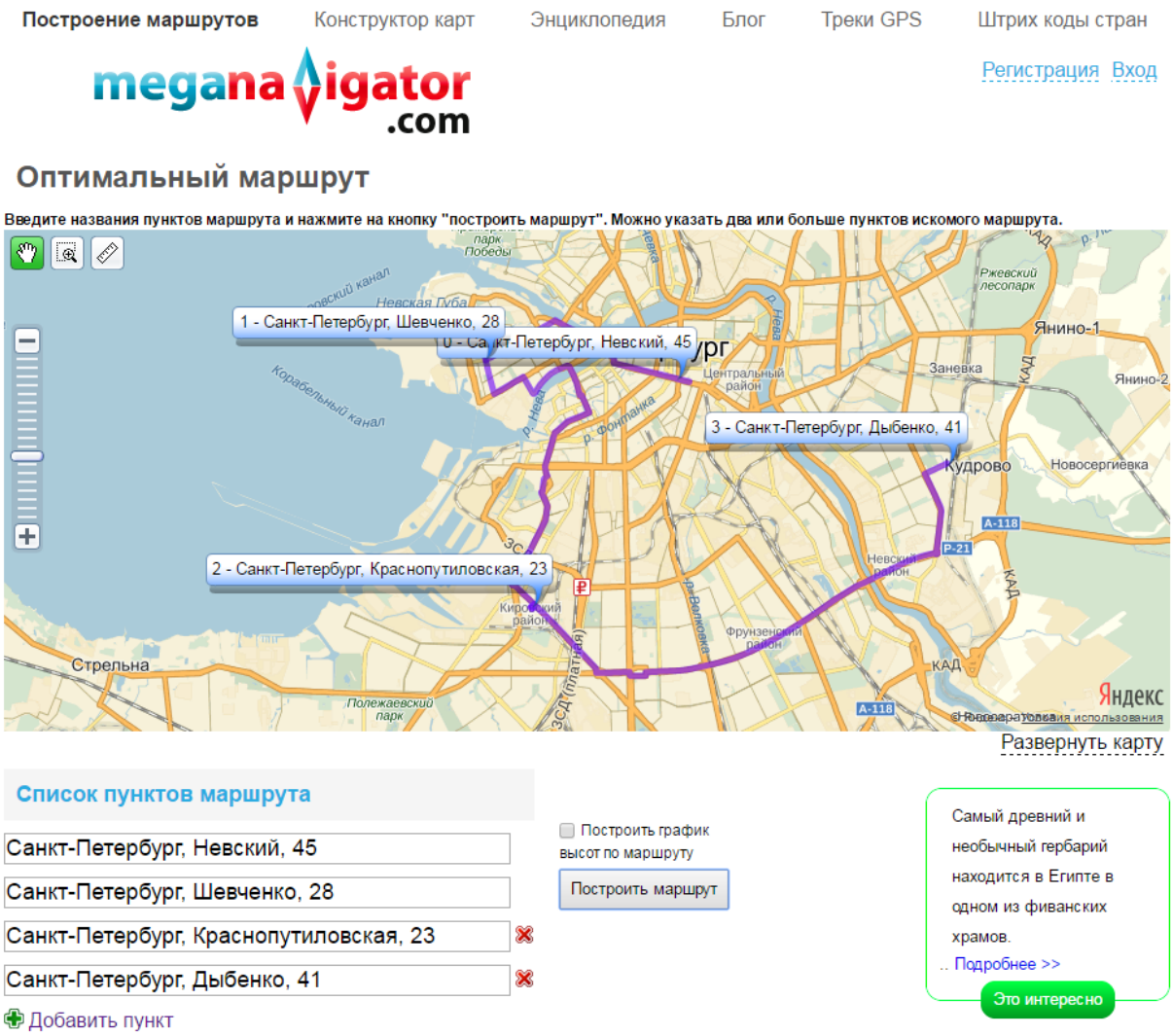

Рисунок 1.1 Сервис MegaNavigator.

Следующий русскоязычный сервис Логист [11] обладает меньшим количеством недостатков, но все они существенны:

1. среди средства передвижения возможен выбор между автомобилем (построение маршрута по проезжим дорогам) и вертолетом (прямой путь между пунктами маршрута);

2. сервис в первую очередь рассчитан для решения задач логистики и построения маршрута перевозок продукции между городами (Рис. 1.2).

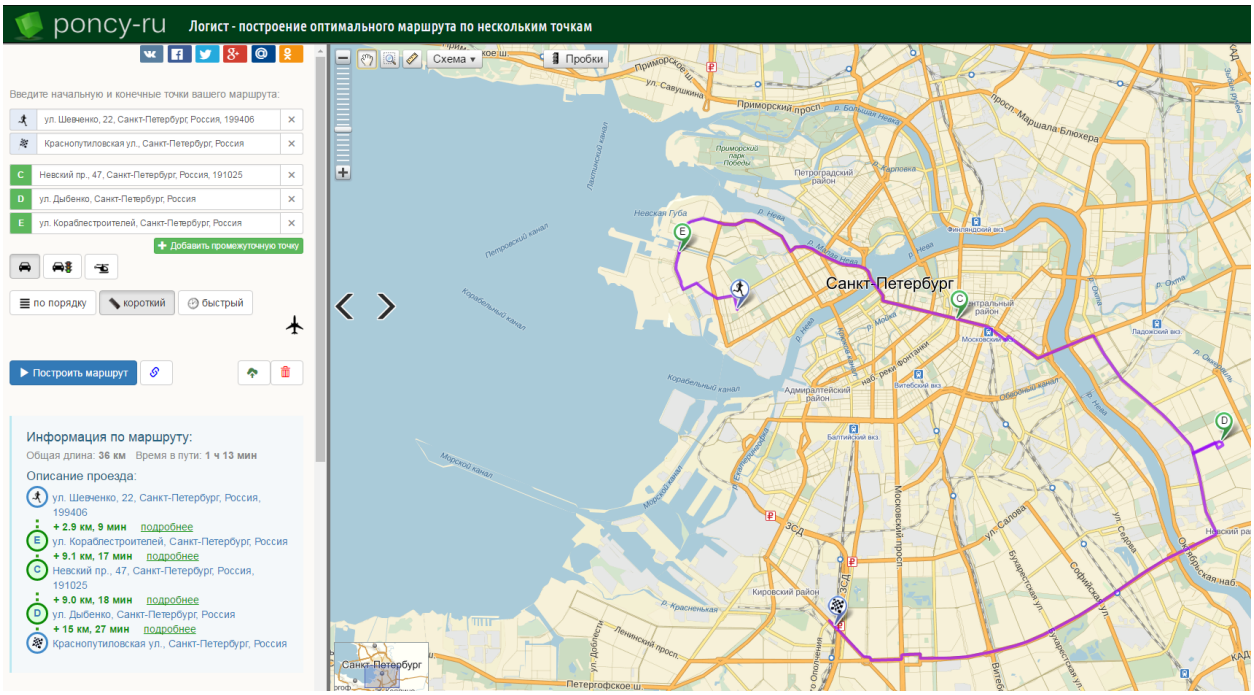

Рисунок 1.2 Сервис Логист.

Среди зарубежных картографических сервисов с функцией построения оптимального маршрута наиболее популярен Speedy Route [12]. В описании продукта сказано, что сервис рассчитывает наиболее эффективный по количеству затрат топлива маршрут между несколькими локациями, где исходная и конечная точка едины (Рис. 1.3). В первую очередь можно выделить такие недостатки как:

1. Speedy Route рассчитан исключительно для автомобилистов;
2. при вводе данных возникают трудности перевода русскоязычных названий на английский язык;
3. у сервиса предусмотрено минимальное количество пунктов маршрута – 5; маршруты, состоящие из 4 пунктов построить невозможно;

4. на сайте представлена урезанная версия карты для ознакомления, для доступа к полной версии требуется оформить платную подписку.

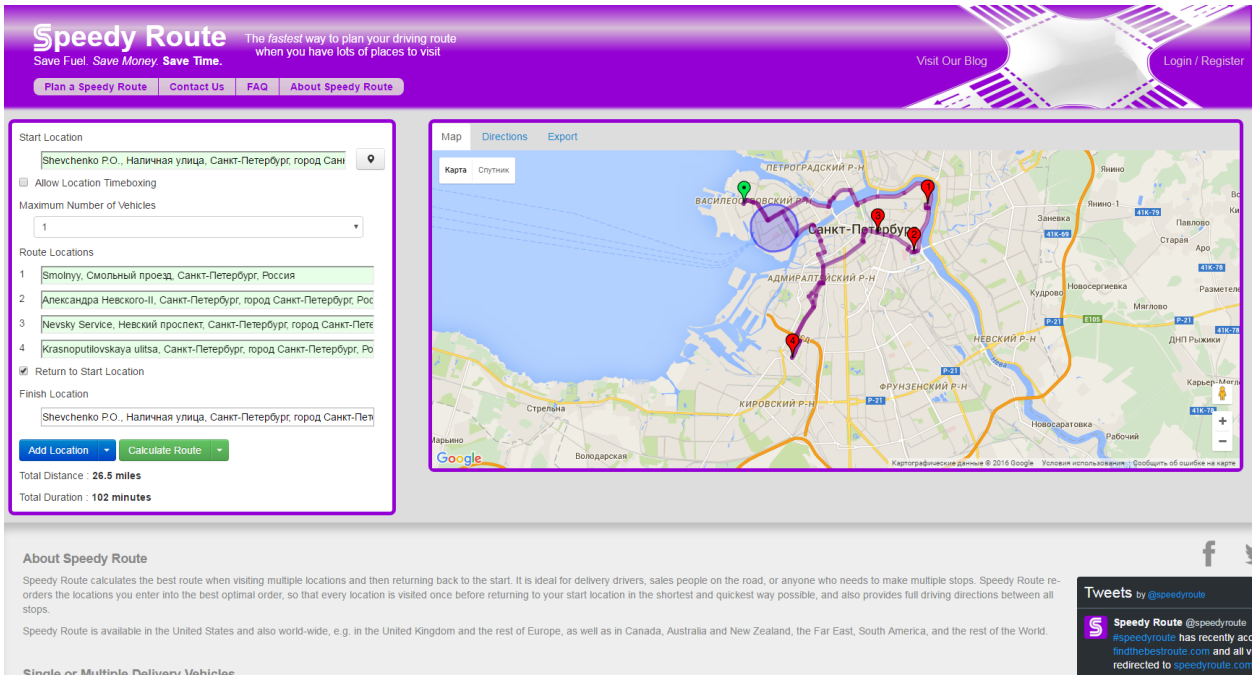

Рисунок 1.3 Сервис Speedy Route.

Суммируя все вышеперечисленное можно сказать, что пока не существуют сервисы, которые бы строили оптимальные пешеходные маршруты внутри города и одновременно были удобными и понятными для пользователей при взаимодействии с ними.

1.3. Постановка задачи

Анализ уже существующих приложений, представленный в предыдущей главе, помогает сформулировать требования, которые можно предъявить к разрабатываемому приложению по оптимальному маршруту между достопримечательностями Петербурга:

1. Функционал приложения заключается в построении оптимального маршрута по заданным данным и выводе его на интерактивной карте;

2. Сервис должен соответствовать законам UX (User Experience) и быть удобным для использования;
3. Приложение будет реализовано на английском языке для расширения аудитории веб-сервиса, в дальнейшем могут быть добавлены другие языки;
4. Одно из ключевых параметров сервиса это его доступность - наиболее удачным решением будет реализация приложения в качестве веб-сайта;
5. В связи со стремительностью развития науки и постоянным нахождением новых методов и оптимизации старых, сервис должен быть спроектирован таким образом, чтобы объем работ по изменению оптимизационного алгоритма был минимален.

1.4. Выводы по Главе 1

В данной главе был представлен обзор предметной области - представлены описание и краткая история формирования задачи поиска оптимального маршрута, приведен анализ недостатков существующих приложений. По результатам обзора была поставлена задача диплома с требования к ее реализации.

ГЛАВА 2. МАТЕМАТИЧЕСКАЯ ОСНОВА ПРОЕКТА

В этой главе будет произведен анализ существующих алгоритмов TSP с описанием их достоинств и недостатков, будет детально разобрано применение генетических алгоритмов к решению задачи поиска оптимального маршрута и показано, почему именно генетический алгоритм был выбран в качестве инструмента разработки приложения.

2.1. Анализ алгоритмов поиска оптимального пути.

Алгоритмы для решения задачи коммивояжера можно разделить на точные (exact algorithm) и неточные (non-exact algorithm). Точные алгоритмы включают в себя перебор всех возможных вариантов, в частных случаях решения могут быть быстро найдены, но в целом осуществляется перебор $n!$ циклов. Вторые в общих случаях применяются для задач, которые невозможно решить точно (вычисление определенных интегралов, решение нелинейных уравнений, извлечение квадратного корня…), если существующие точные решения требуют значительных и неоправданных временных затрат при высокой сложности задачи, и как часть более сложного алгоритма, с помощью которого задача решается точно. [13]

2.1.1. Точные алгоритмы

В свою очередь существует две группы точных алгоритмов — одна из них использует методы релаксации линейного программирования TSP: алгоритм Гомори, метод внутренней точки, метод ветвей и границ; вторая, меньшая группа, использует методы динамического программирования. Характерная особенность методов обеих групп — гарантия нахождения оптимальных решений при общей трудоемкости процесса. [14]

Полный перебор (Brute Force)

Один из самых очевидных методов решения задачи коммивояжера – метод полного перебора или грубой силы. Его суть заключается в переборе всех возможных вариантов путей, алгоритм решения можно записать как:

1. найти общее число возможных гамильтоновых контуров;
2. найти вес каждого гамильтонова контура, сложив вес всех его ребер;
3. выбрать гамильтонов контур с минимальным весом, который и будет оптимальным.

Метод полного перебора обладает рядом преимуществ – он гарантирует нахождение решения задачи TSP, при этом он прямолинеен и прост в исполнении. В то же время, алгоритм считается неэффективным при работе с большим объемом данных, так как для нахождения оптимального маршрута требует найти вес $(n - 1)!$ гамильтоновых контуров.

Таблица 2.1 демонстрирует количество времени, требуемое для решения задачи коммивояжера методом полного перебора при компьютерной мощности, позволяющей считать 1 миллион гамильтоновых контуров в секунду.

Расчетное время решения TSP методом полного перебора	
Количество городов	**Расчетное время**
10	1/3 секунды
13	8 минут
15	1 год
20	193 года

Таблица 2.1 Расчетное время решения TSP методом полного перебора. [15]

Метод ветвей и границ (Branch and Bound)

Метод ветвей и границ часто используется для нахождения оптимального решения задач комбинаторной оптимизации. Его суть заключается в разбитии множества на подзадачи и исключении заведомо неоптимальных решений.

Пусть граф V содержит все города, $П$ – множество всех перестановок городов, покрывающее все возможные решения. Рассмотрим перестановку $\pi \in П$, в которой каждому городу назначается преемник – i для πi города. Таким образом, тур можно записать как $(1, \pi(1), \pi(\pi(1)), \dots, 1)$. Если число городов в туре равно n, тогда перестановку называют циклической. Задача о назначениях ставит перед собой цель найти циклические перестановки, а задача коммивояжера преследует ту же цель, но с ограничением, что у этих перестановок должна быть минимальная стоимость. Метод ветвей и границ в первую очередь находит решение задачи о назначениях, стоимость которой для n городов довольно большая и асимптотически равна $O(n^3)$. [16]

Если был найден полный тур, то полученное значение также является решением задачи коммивояжера. В противном случае проблема разделяется на несколько подобластей, каждая из которых исключает некоторые дуги подтура, таким образом исключая сам подтур. Метод, с помощью которого высчитывается, какую дугу следует удалить, называют правилом ветвления. Важное замечание – не должно существовать дублированных подзадач, их общее количество должно быть минимизировано.

Будем использовать критерий, гарантирующий независимость подзадач – рассматривается включенный набор дуги и выбирается минимальное число дуг, которые не принадлежат набору. Обозначим E как множество исключенных дуг и I как множество включенных. Разложим I. Выберем t дуги подобластей $x_1 x_2 \dots x_n$ которые не принадлежат I. Задача разделена на t

потомков так, чтобы у j_{th} потомка были E_j исключенных дуг и I_j включенных дуг. Запишем в виде формулы:

$$\left.\begin{array}{c} E_j = E \cup \{x_j\} \\ I_j - I \cup \{x_1, x_2, \dots x_{j-1}\} \end{array}\right\} \mathrm{k} = 1, 2, \dots . \mathrm{j}$$

Но x_j - исключенная дуга j_{th} подзадач и включенная дуга в $(j+1)_{st}$ области. Это значит тур, полученный решением $(j+1)_{st}$ задачи, может иметь x_j дугу, но тур, полученный решением $(j+1)_{st}$, не содержит эту дугу. Это гарантирует отсутствие дублирующихся маршрутов.

Количество возможных решений равно $(n-1)!/2$, для n=50 это приблизительно 3×1062. Этот метод наиболее часто используется при количестве узлов от 40 до 60. [17]

Необходимость целиком решать задачи линейного программирования во всей области допустимых решений можно считать главным недостатком вышеописанного метода. Для задач с большим объемом данных метод ветвей и границ является неоправданно трудоемким, в то же время алгоритм является надежным методом решения целочисленных задач.

Алгоритм Гомори (The Cutting Plane)

В 1954 году была представлена работа Данцига, Фалкерсона и Джонсон, описывающая новый метод решения задачи коммивояжера, который также может быть использован для решения любой проблемы

$$minimize\ c^T x\ subject\ to\ x \in S$$

где $c \neq 0$, S – конечное подмножество некоторого R^m, и таким образом это мы сможем найти точки S. Это итерационный алгоритм – каждое повторение начинается с линейной программной релаксации. Запишем в виде формулы:

$$minimize\ c^T x\ \ subject\ to\ Ax \leq b$$

где многогранник P, определенный как $\{x : Ax \leq b\}$, содержит S и ограничен. Так как P ограничен, мы можем найти оптимальное решение x^* как экстремальную точку P. Если x^* принадлежит S, то оптимальное решение найдено (2.2); в противном случае некоторое линейное неравенство удовлетворяет все точки S и нарушает x^*. Такое неравенство называют алгоритмом Гомори, который подробно описал его 1958, методом отсекающих плоскостей или просто отсечений. [18]

Данный метод используется для построения точных или приближенных задач, особенно часто встречается в сочетании с методом ветвей и границ и тогда называется методом ветвей и отсечений. Оба метода основаны на решении последовательности релаксированных подзадач линейного программирования. В алгоритме Гомори релаксированные подзадачи постепенно улучшают аппроксимацию целочисленной задачи, уменьшая окрестность оптимального решения. Если оптимальность не удалось получить, тогда ищется приближенное решение с погрешностью.

У метода отсечений есть преимущество над методом ветвей и границ – первые более удобны для аппаратного вычисления, так как для их решения не требуется большой объем оперативной памяти для хранения дерева решений. [19]

Метод динамического программирования (Dynamic Programming)

Рассмотрим задачу с n городами и расстояниями d_{ij} между любыми двумя городами, путь начинается и заканчивается в городе n_0. TSP может быть решена за время $O(n!)$ методом полного перебора, но алгоритм динамического программирования позволяет сократить то время до $O(n^2 2^n)$.

Обозначим подзадачу: пусть S будет подмножеством городов, содержащим 1 и как минимум еще один город, j будет городом отличным от 1, обозначим через $C(S, j)$ самый короткий путь, начинающийся в 1,

проходящий все города S и заканчивающийся в j. Запишем алгоритм в виде псевдо-кода.

$$for\ all\ j\ do\ C(\{1, j\}, j) := d_{1j}$$

$$for\ s := 3\ to\ n\ do\ (the\ size\ of\ the\ subsets\ considered\ this\ round)$$

$$for\ all\ subsets\ S\ of\ \{1, \dots n\} of\ size\ n\ and\ containing\ 1\ do$$

$$for\ all\ j \in S, j \neq 1\ do$$

$$C(S;\ j) := min_{i \neq j, i \in S}[C(S - \{j\}, i) + d_{ij})]$$

$$opt := min_{j \neq 1}[C(\{1, 2, \dots n\}, j) + d_{j1}]\ [20]$$

Данный метод используется для повышения эффективности вычислительных повторений, храня промежуточные результаты и снова используя их при необходимости.

2.1.2. Неточные алгоритмы

В целом алгоритмы данной группы предлагают потенциально неоптимальные, но быстрые решения. В свою очередь приближенные алгоритмы можно разделить на две категории: приближенные (Approximation Algorithms) и эвристические (Heuristic Algorithms).

Алгоритм Кристофидеса (Christofides' Algorithm)

Алогоритм Кристофидеса используется для решения метрических TSP – с дополнительным условием, что для матрицы расстояний выполнено неравенство треугольника:

$$\forall i, j, k\ \ d_{ik} \geqslant d_{ij} + d_{jk}$$

Большая часть эвристических алгоритмов относятся к 2-приближенному классу. Профессор Никос Кристофайдс в 1976 году доработал один из существующих алгоритмов (метод двойного минимального остовного дерева,

$O(n^2 log_2(n)))$ так, что время решения задачи не превышает оптимальное время более чем на 3/2. [21]

Решение оригинального алгоритма можно записать так: найти минимальное дерево из множества всех городов, продублировать все ребра и построить эйлеров граф, построить гамильтонов цикл, пройдя каждый узел только один раз и выбирая наикратчайших путь, ведущий из каждого узла.

Алгоритм Кристофидеса состоит из последовательности следующих действий:

1. найти минимальное дерево из множества всех городов;
2. найти паросочетание с минимальным весом множества вершин нечетной степени и построить эйлеров граф;
3. найти эйлеров обход и построить гамильтонов цикл, избегая посещаемых узлов. [21]

Основное различие – дополнительное вычисление паросочетания с минимальным весом. Эта часть также самая трудоемкая, поэтому время выполнения алгоритма возрастает до $O(n^3)$. Проведенные тесты показали, что алгоритм Кристофидеса на 10% выше нижней границы Хелд-Карп. [22]

Алгоритм ближайшего соседа (NearestNeighbour)

Один из самых простых эвристических методов решения TSP. Главное правило алгоритма – всегда выбирать близлежащий город (соседа). Решение задачи складывается из следующих шагов:

1. выбрать любой город;
2. найти близлежащий город, не включенный в маршрут, и перейти в него;
3. проверить остались ли города, не включенные в маршрут, если ответ положительный – повторить второй шаг.

4. чтобы завершить тур добавить ребро между последним выбранным городом и первым.

В общем случае трудоемкость решения задачи равна O(n^2). Нижняя граница стоимости оптимального маршрута на 10% выше нижней границы Хелд-Карп. [23]

Жадный алгоритм (Greedy)

Чтобы решить TSP использование жадный алгоритм, мы исследуем все ребра, выходящие из города-узла, и выбираем n самых коротких дуг. Если те n самых коротких дуг формируют гамильтонов цикл, тогда мы нашли оптимальное решение. [24]

Трудоемкость решения задачи жадным алгоритмом равна O(n^2). Нижняя граница стоимости оптимального маршрута выше нижней границы Хелд-Карп на 15-20%. [22]

Алгоритм Кернигана – Лина (Lin-Kernighan)

Алгоритм Кернигана – Лина считается один из самых эффективных методов поиска оптимальных или почти оптимальных решений задачи коммивояжера. Однако разработка и реализация алгоритма не проста, так как алгоритм состоит из множества шагов, большинство из которых сильно влияет на работу алгоритма. [25] Создание алгоритма Кернигана было вдохновлено наблюдением, что статическое K в -оптимальном методе не дает наилучшее решение. Появилась идея использовать различные стадии -оптимального метода в выполнении эвристического алгоритма. На практике было показано, что практически невозможно заранее предугадать какое K следуюет использовать, чтобы достигнуть лучшего компромисса между трудоемкостью и качеством решения. Лин и Керниган убрали этот недостаток, введя

оптимальную переменную, таким образом значение K меняется во время выполнения алгоритма. [26] Трудоемкость при этом равна O($n^{2.2}$). [22]

Алгоритм поиска с запретами (TabuSearch)

Главная проблема алгоритма ближайшего соседа состоит в частом застревании в точке локального оптимума. Этого можно избежать, применив алгоритм поиска с запретами, в 1977 году предложенный Ф. Гловером. Данный метод позволяет переходить от одного локального оптимума к другому в поиске глобального оптимума, после перехода ребро попадает в список запретов и повторно не используется, кроме случаев, когда оно может улучшить построенный оптимальный путь. На практическом уровне запрещенный набор сохраняется как комбинация ранее посещаемых шагов, который позволяет построить дальнейший путь относительно текущего решения и соседних узлов. [27]

Главным недостатком этого метода служит его время выполнения – трудоемкость алгоритма оценивается как $O(n^3)$. [22]

Муравьиный алгоритм (Ant Colony Optimization)

Муравьиный алгоритм – эффективный полиномиальных алгоритм, вдохновленный вдохновлена поведением настоящих муравьев. Впервые его принципы были описаны в 1991 Марко Дориго. Муравьям свойственно сотрудничать в поисках пищевые ресурсы, поэтому они оставляют след химического вещества, феромонов, на их пути от гнезда до источника пищи. [26] Этот тип невербальной коммуникации называют стигмергия – стимуляция, основанная на опыте предыдущих муравьев и направленная на повышение производительности. [28]

Для решения задачи коммивояжера как правило используют около 20 муравьёв. Их размещают в случайные города и отправляют в другие города. Им не позволяют дважды посещать один и тот же город, только если они не завершают маршрут. Тот муравей, который выбрал самый короткий тур, будет оставлять след феромонов обратно пропорциональный длине маршрута. Этот след феромонов будет считываться следующим муравьем при выборе города, и с большой вероятность он пойдет тем же путем, еще сильнее укрепив след. Этот процесс будет многократно повторен пока не будет найдет маршрут, достаточно короткий, чтобы быть оптимальным.

Среди недостатков алгоритма хочется выделить, что первое полученное решение может оказаться одним из худших в плане оптимизации, однако при повторном решении метод выдает достаточно точный результат. [29]

Нижняя грань Хелд-Карпа (The Held-Karp Lower Bound)

Самый распространенный способ измерить эффективность эвристического алгоритма для решения TSP – это сравнить результаты с нижней гранью Хелд-Карпа. Эта нижняя грань является решением TSP, найденным за полиномиальное время при помощи симплекс-метода. Нижняя грань Хелд-Карпа приблизительно 0.8% ниже оптимальной продолжительности тура. [30] В то же время она гарантировано не превышает оптимальное время более чем на 2/3. [22]

По состоянию на 2015 алгоритм Кристофидеса считается самым эффективных методом для решения задачи коммивояжера на общих метрических пространствах, хотя известны лучшие приближения для частных случаев. Также хорошо в тестах себя показали алгоритм Кернигана-Лина и жадный эвристический алгоритм. [22]

2.2. Генетический алгоритм

Генетические алгоритмы принадлежат к методам оптимизации, в основу которых легли биологические процессы, протекающие в природе. Чарльз Дарвин в своей эволюционной теории ввел определение естественного отбора, согласно которой особи, более присоленные к условиям окружающей среды, имеют больше шансов на выживание и продолжение рода, и наоборот – неприспособленные осыпи подвергаются избирательному уничтожению. Основой отбора являются мутации генов и их комбинации, формирующиеся при размножении и передающиеся потомству. В ходе естественного отбора выживают экземпляры с наибольшей функцией приспособленности. Это численная характеристика, которая может изменяться в зависимости от условий конкретной задачи. Приспособленные особи скрещиваются (кроссовер) и производят потомство. Случайные мутации также могут оказывать влияние на развитие популяции. На рисунке 2.2. представлена общая схема генетического алгоритма: [19]

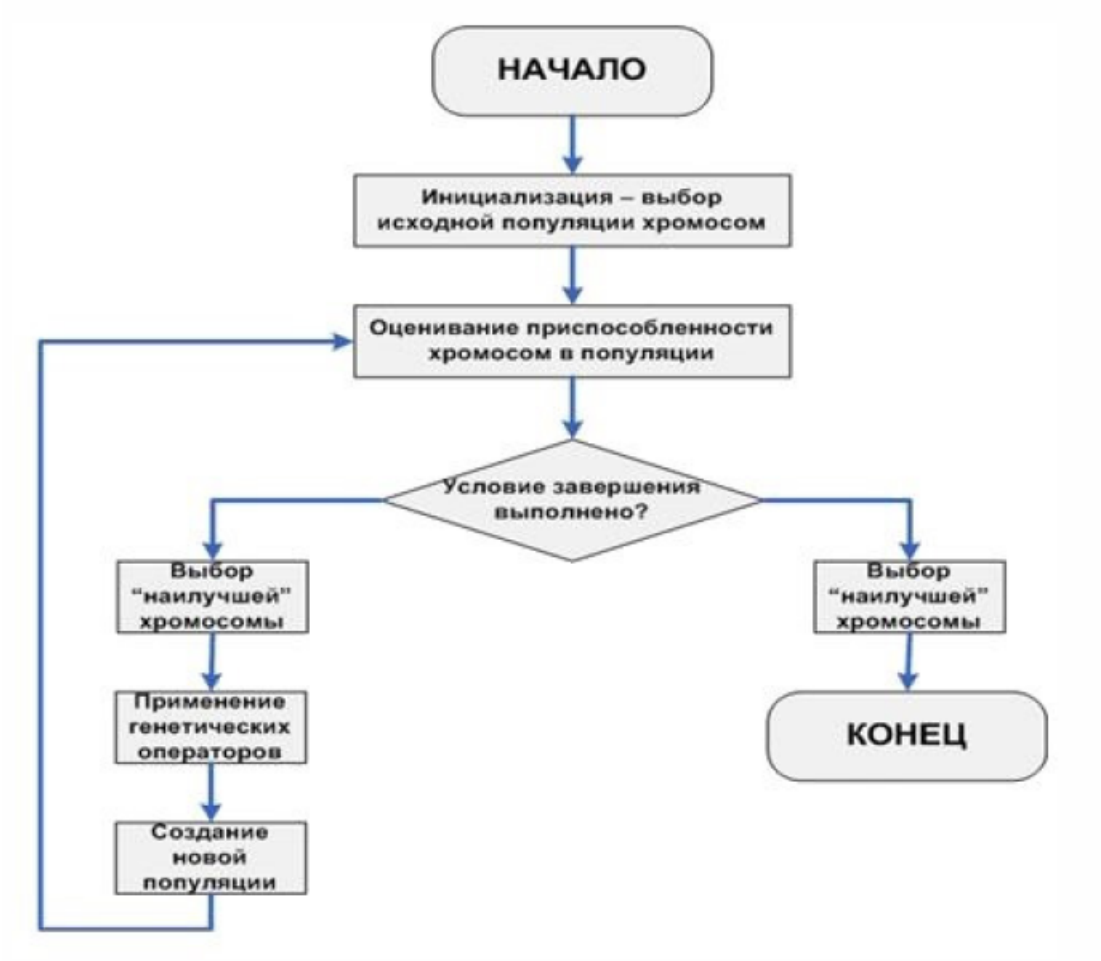

Рисунок 2.1 Общая схема генетического алгоритма [19]

Мы можем обозначить задачу оптимизации как задачу нахождения функции $f(x_1, x_2, \ldots, x_n)$, носящей название функция приспособленности, которая позволяет выделить наиболее приспособленных особей популяции для размножения и наименее приспособленных, которые вычеркиваются, повышая тем самым приспособленность нового поколения. Требуется, чтобы на области определения выполнялось неравенство $f(x\_1, x\_2, \ldots, x\_n) \geq 0$, область является ограниченной. Параметры функции записывается как строки, состоящие из битов. Строка, полученная конкатенацией строк, называется особью: [19]

$$1010\ 10110\ 101 \ldots 10101$$

$$| x1 | x2 | x3 | \ldots | xn |$$

Генетический алгоритм универсален, так как только функция приспособленности и кодирование решений зависят от условий поставленной задачи. При выполнении алгоритма учитываются следующие правила: начальная популяция выбирается случайно, количество ее особей остается неизменным, каждая из них записывается как строка с определенной длиной кодировки. [19]

Каждый шаг алгоритма можно разбить на три этапа:

1) пропорциональный в зависимости от приспособленности отбор особей текущего поколения, имеющих право производить потомство, и формирование из них промежуточной популяции;

2) промежуточную популяцию делят на пару и с какой-то вероятностью скрещивают, в итоге в новое поколение попадает сама пара или ее потомки при их присутствии. Потомки формируются из отсеченных частей родительских строк, разделенной некой точкой.

3) происходит мутация полученного поколения, не допускающая преждевременной сходимости. Каждый бит особи с вероятностью не более 1% записывается как противоположный исходному. [19]

В качестве заранее заданных критериев получения оптимального решениях могут быть определенное число смены поколений или схождения популяции – условие выполняется, если все строки являются почти идентичными и находятся в области экстремума. Решением алгоритма будет особь с наибольшим значением функции приспособленности. [19]

При использовании генетического алгоритма для решения задачи поиска оптимального пути выполняются все вышеперечисленные шаги, приспособленность особи фактически служит мерой длины маршрута. Существуют несколько различных реализаций классического генетического алгоритма в зависимости от подхода к этапам скрещивания и мутации. Некоторые из них дали хорошие показатели при тестировании, сильно превзойдя алгоритм Кернигана-Лина. Но также, как и у алгоритма поиска с

запретами трудоемкость процесса создает проблему при достаточно большом количестве узлов. [19] Исследование, представленное в статье “Comparison of Algorithms for Solving Traveling Salesman Problem” [31] показывает результаты сравнения трех методов: алгоритма ближайшего соседа, генетического и жадного алгоритмов. В качестве примера используется карта Соединенных Штатов Америки, областью 9.9 миллионов км., представленная на Рисунке 2.2. Города были выбраны произвольно.

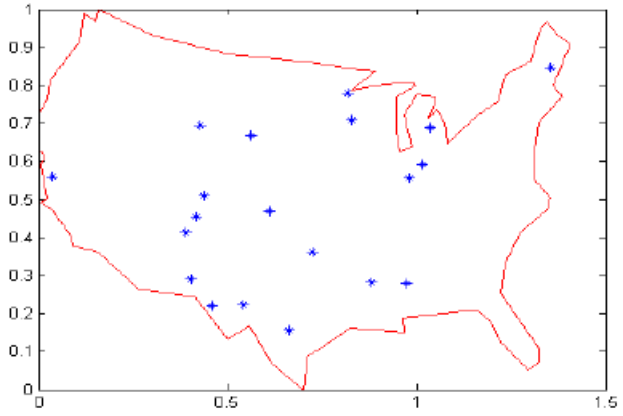

Рисунок 2.2 Исходный пример из 20 городов [31]

Оптимальное решение показано на Рис. 2.3 с общей длиной маршрута 4616 км. Это решение имеет высокую сложность и состоит из наибольшего числа итераций.

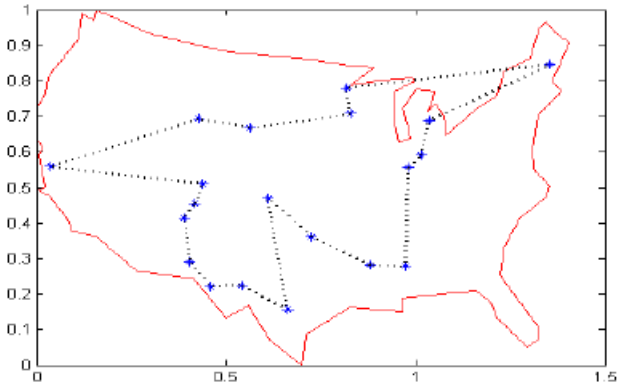

Рисунок 2.3 Исходный пример из 20 городов [31]

Решая ту же задачу методом ближайшего соседа, был получен маршрут, показанный на рисунке 2.4. Его длина составляет 15800км.

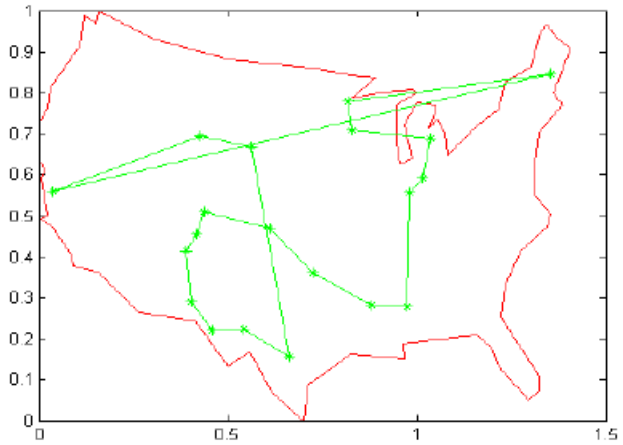

Рисунок 2.4 Маршрут, полученный с использованием алгоритма ближайшего соседа [31]

Решение TSP для 20 городов с использованием генетического алгоритма показано на рисунке 2.5. Хотя было совершено больше итераций, чем в предыдущем примере, данный алгоритм нашел более оптимальный маршрут равный 11900км.

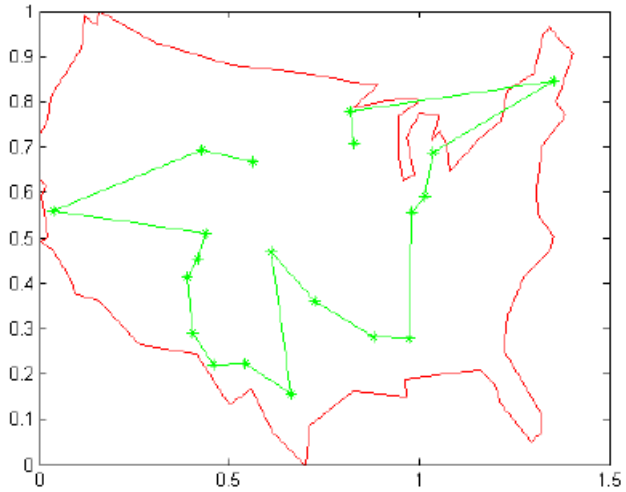

Рисунок 2.5 Маршрут, найденный генетическим алгоритмом [31]

Тот же самый пример TSP был решен с использованием жадного алгоритма. Результат показан на Рисунке 2.6. и имеет длину 12900км. Хотя это решение имеет незначительно сокращение расстояния по сравнению с

алгоритмом ближайшего соседа, у него все же более высокая сложность и время выполнения. Генетический алгоритм показал себя как самый эффективный метод решения TSP с небольшим объемом данных, но в то же время как самый трудоемкий.

Подобные сравнения были приведены для задач большой размерности. Результаты исследования, проведенные на 2,2 GHz 16GB of 1600MHz DDR3L SDRAM Intel Core i7 MacBook и включающие в себя сравнение таких параметров как длина оптимально пути в километрах, затраченное время в секундах и количество итераций, представлены на Таблице 3.1.

Сравнение алгоритмов для решения TSP при 100 городах			
Выбранный алгоритм	Длина оптимального маршрута (км)	Затраченное время (сек)	Количество итераций
Алгоритм ближайшего соседа	26664	2.5	100
Генетический алгоритм	225479	45	10000
Жадный алгоритм	23311	0.07	18
Сравнение алгоритмов для решения TSP при 1000 городах			
Выбранный алгоритм	Длина оптимального маршрута (км)	Затраченное время (сек)	Количество итераций
Алгоритм ближайшего соседа	83938	95.5	1000
Генетический алгоритм	282866	468	10000
Жадный алгоритм	72801	127	151

Таблица 3.1 Сравнение алгоритма ближайшего соседа, генетического и жадных алгоритмов при решении задач с 100 и 1000 городами. [31]

Рисунок выше наглядно показывает, что при количестве городов в задаче коммивояжера близком к 100, генетический алгоритм проигрывает жадному по всем параметрам. При анализе TSP с 1000 исходными городами генетический алгоритм в корне неэффективен. [31]

Суммируя сказанное выше, можно сказать, что главным недостатком генетического алгоритма является отсутствие гарантии, что полученное решение является оптимальный, как и само его нахождение за приемлемое оптимально время.

В то же время генетические алгоритмы показывает высокую эффективность в задачах с небольшим количеством городов. В ситуациях, когда в задачах большой размерности отсутствует упорядоченность вводных данных, генетический алгоритм является единственной альтернативой алгоритму полного перебора. Главное его преимущество – его можно использовать для решения сложных неформализованных проблем для которых не существует частных методов решения, что позволяет ему эффективно решать нестандартные задачи. [19]

2.3. Выводы по Главе 2

В главе 2 был проведен анализ методов решения TSP, представлено краткое описание алгоритмов, перечислены их достоинства и недостатки. Более детально был описан генетический алгоритм, также подведены итоги сравнительного исследования алгоритма с рядом другим. По результатам данного сравнения было выявлено, что для задач с небольшим количеством городов, к которым относится практическая задача дипломной работы, одним из наиболее эффективных является генетический алгоритм, который и был выбран для дальнейшей работы.

ГЛАВА 3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ И РЕЗУЛЬТАТЫ РАБОТЫ

Третья глава описывает используемые для разработки приложения инструменты, также приводится алгоритм, строящий оптимальный пешеходный маршрут по заданным на карте точкам. В конце главы дается анализ работы приложения.

3.1. Обзор используемых в разработке приложения инструментов

Приложение реализовано исключительно на клиентской стороне, ниже приведены инструменты, используемые при написании, такие как HTML5, CSS3, фреймворк Bootstrap 3, JavaScript и библиотека jQueary, Goggle Maps, WebGL.

3.1.1. HTML, CSS и Bootstrap

HyperText Markup Language (HTML) – язык разметки WorldWideWeb для создания веб-страниц. [32] В приложении используется новейший на сегодняшний день стандарт HTML5, вышедший в 2014 году. Главные его отличия от предыдущей версии это расширенная поддержка мультимедиа и ввод дополнительных элементов. [33]

Cascading Style Sheets (CSS) – язык разметки, отвечающий за внешний вид веб-страницы, написанный в виде каскадных таблиц стилей. В коде был использован стандарт CSS3, поддерживающий анимацию, сглаживания, градиенты и различные другие особенности. [34] При написании CSS также был задействован препроцессор Stylus, сильно упрощающий синтаксис CSS и позволяющий использовать переменные и функции, недоступные в исходном языке. [35]

Для простоты работы с адаптацией приложения для экранов планшетов и мобильных телефонов, а также повышения кроссбраузерности был выбран фреймворк Bootstrap 3. [36]

3.1.2. JavaScript и jQuery

Приложение работает на клиентской стороне, в связи с этим в качестве ведущего языка был выбран JavaScript. На нем написан как генетический алгоритм, так и вывод полученного оптимального пути непосредственно на самой карте после того как пользователь выберет точки маршрута.

JavaScript (JS) – объектно-ориентированный язык сценариев веб-страниц. Код, написанный на JavaScript, встраивается в исходную HTML код и считывается браузером по мере загрузки веб-страницы. С его помощью можно динамически изменять как HTML так и CSS код страницы в зависимости от действий посетителей сайта или приложения. [37]

Кроме чистого JS в приложении используется библиотека jQuery, упрощающая взаимодействие с деревом DOM – объектной модели HTML-документов. [38]

3.1.3. Google.Maps

Google Maps – это веб-сервис, который предоставляет подробную информацию о географических регионах и объектах по всему миру. Сервис позволяет выбрать между стандартным дорожным, спутниковым и гибридным режимами просмотра карт. В некоторых городах также представлена функция Google Street View со стереосферическими панорамами. Планировщик маршрутов предлагает маршруты для водителей, велосипедистов, пешеходов и передвигающихся общественным транспортом. Сервис Google Maps для мобильных устройств предоставляет возможность определения

местоположения для автомобилистов, использующих систему глобального позиционирования (GPS) местоположение мобильного устройства вместе с данными из беспроводных и сотовых сетей. В качестве бонуса система также предлагает карты Луны, Марсы для любителей астрономии. Одной из наиболее значимых разработок было создание открытого Google Maps API, который позволяет разработчикам использовать карты для сторонних сервисов и управлять ими с помощью JavaScript. С 2013 года трехмерный спутниковый режим карт работает с использованием библиотеки WebGL. [39]

3.1.4. WebGL

Web-based Graphics Library (WebGL) – это библиотека для языка JavaScript, позволяющая рисовать, отображать и взаимодействовать со сложной, интерактивной трехмерной компьютерной графикой в веб-браузерах.

Первоначально для использования трехмерной графики требовалось наличие высокопроизводительного компьютера или специализированной игровой консоли. Между тем в настоящее время такие ограничения отсутствуют в силу роста производительности персональных компьютеров и развития веб-браузеров. Благодаря этому отображение трехмерной графики стало возможным и с помощью веб-технологий. [40]

До выхода WebGL динамические веб-страницы создавались с использованием HTML и JavaScript, но после его релиза в 2013 году к ним добавился язык шейдеров, компьютерных программ, позволяющих создавать сложные визуальные эффекты с использованием специализированного языка программирования – GLSL ES, что демонстрирует рисунок 3.1. При этом GLSL ES изначально написан на языке JavaScript, из этого следует, что несмотря на то, что WebGL увеличивает сложность сценариев на JavaScript,

структура приложений остается той же – используются только файлы HTML и JavaScript. [40]

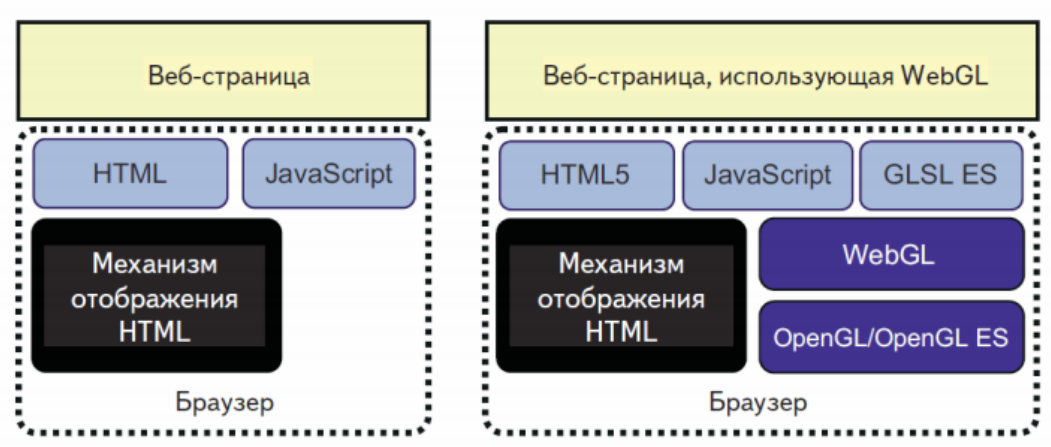

Рисунок 3.1 Структура страницы без и с использованием WebGL. [40]

В приложении WebGL используется как в картах GoogleMaps, о чем было сказано выше, так и в качестве подключаемой библиотеки для работы визуального эффекта дождя на первой секции приложения.

3.2. Логика приложения

Приложение JourneyMap представляет из себя лэндинговую страницу на данный момент доступную по адресу http://journeymapspb.ru.

Приложение полностью адаптировано для планшетов и мобильных телефонов, что позволяет использовать его непосредственно во время экскурсионных прогулок и быстро корректировать маршруты.

Первая секция является вводной и рассказывает посетителю о работе JourneyMap. Визуальный эффект моросящего дождя вписывается в концепт приложения.

Секция представлена на рисунке 3.2.

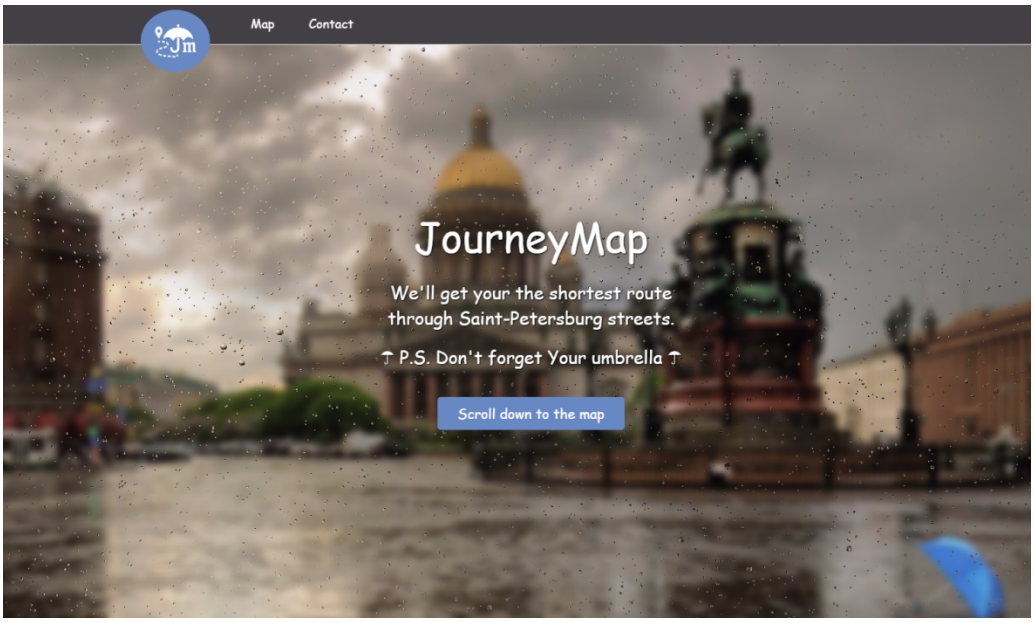

Рисунок 3.2. Вводная секция приложения JourneyMap.

Вторая секция представляет собой встроенную карту, окно с доступной инструкцией по работе с ней и кнопки взаимодействия. По умолчанию Goggle.Maps представлена в режиме “Спутник с названиями объектов” и показывает центр Санкт-Петербурга. Для посетителей, привыкших к стандартным картам, были добавлены кнопки переключения между режимами просмотра. Рисунок 3.3. показывает описанную секцию.

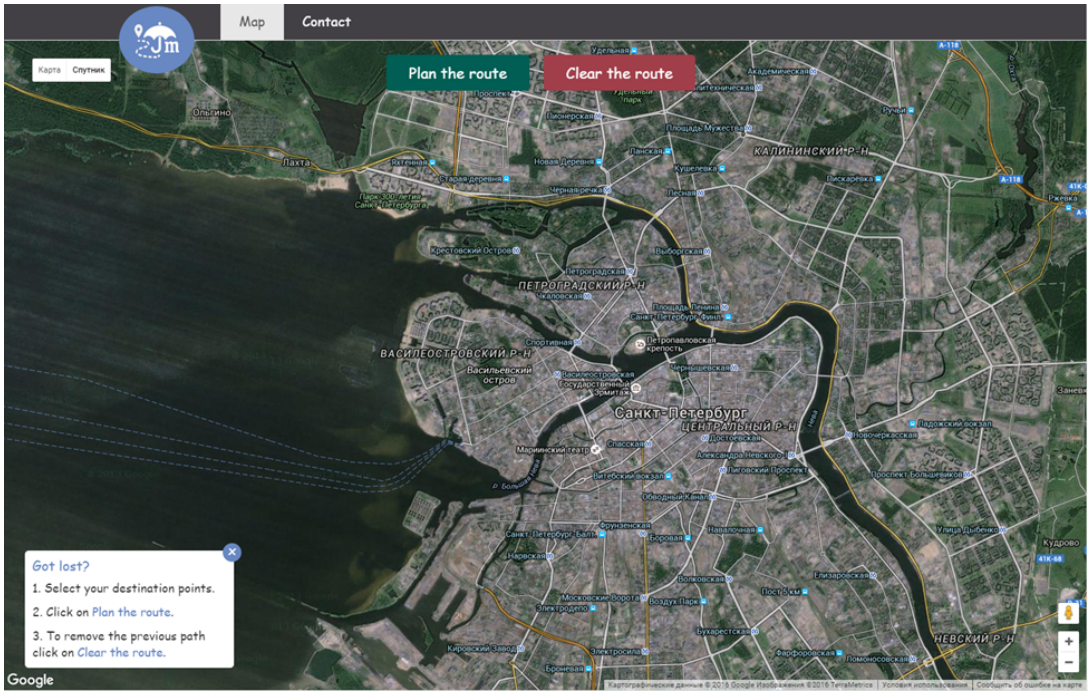

Рисунок 3.3. Вторая секция приложения JourneyMap со встроенной картой.

На рисунке 3.4 пользователь расставляет маркеры на карте, задавая узлы для будущего оптимального маршрута.

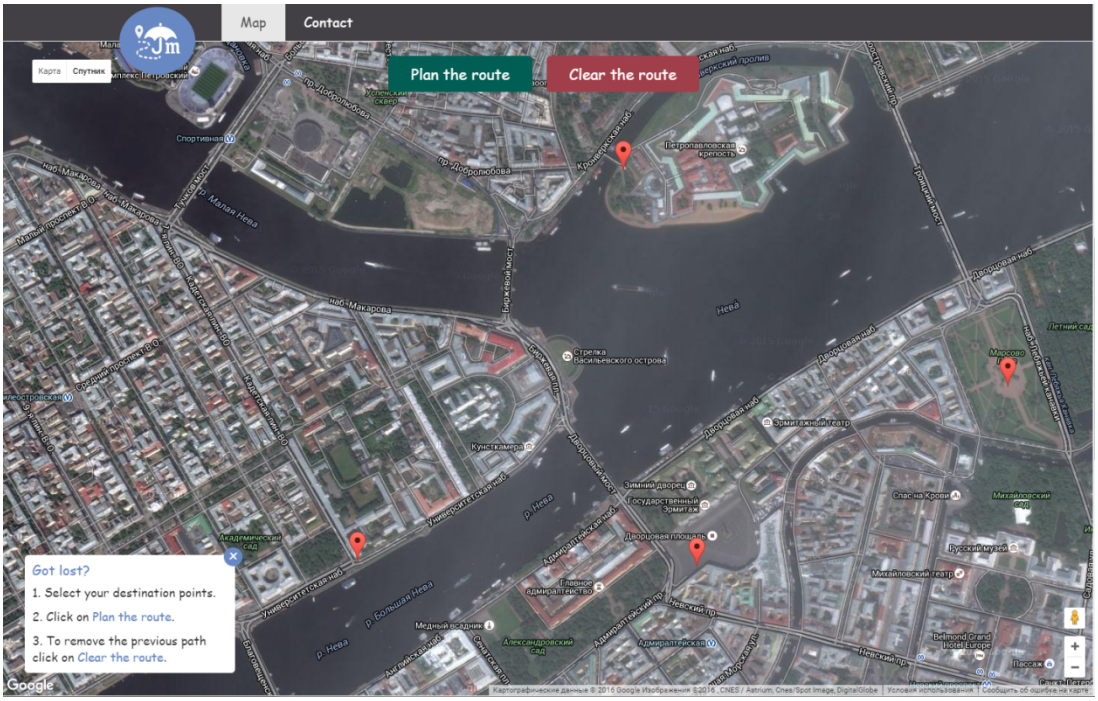

Рисунок 3.4. Вторая секция приложения JourneyMap с выбранными узлами маршрута.

После того, как все узлы были расставлены, посетитель нажимает на кнопку “Plan the route” и ему показывается оптимальный пеший маршрут между заданными достопримечательностями. Порядок представлен латинскими буквами, где исходная и конечная точка одинаковы – в качестве этого узла может быть задана гостиница или квартира путешественника. Кнопка “Clear the route” дает возможность удалить существующий маршрут и ввести новые данные. Рисунок 3.5 отображает описанные действия.

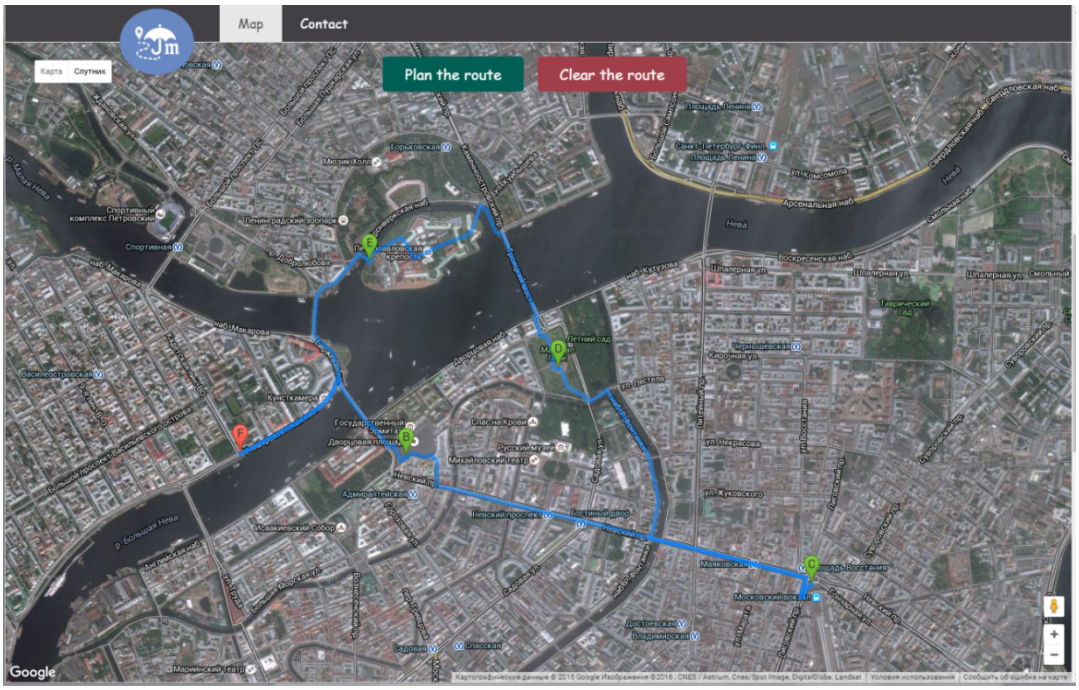

Рисунок 3.5. Вторая секция приложения JourneyMap с построенным маршрутом

На третьей секции Контакты, представленной на рисунке 3.6, посетитель, нажав на кнопку “Email me”, может оставить свой отзыв о приложении или оправить запрос на добавление нового функционала – сервис специально было написан гибким к изменениям для его последующего расширения.

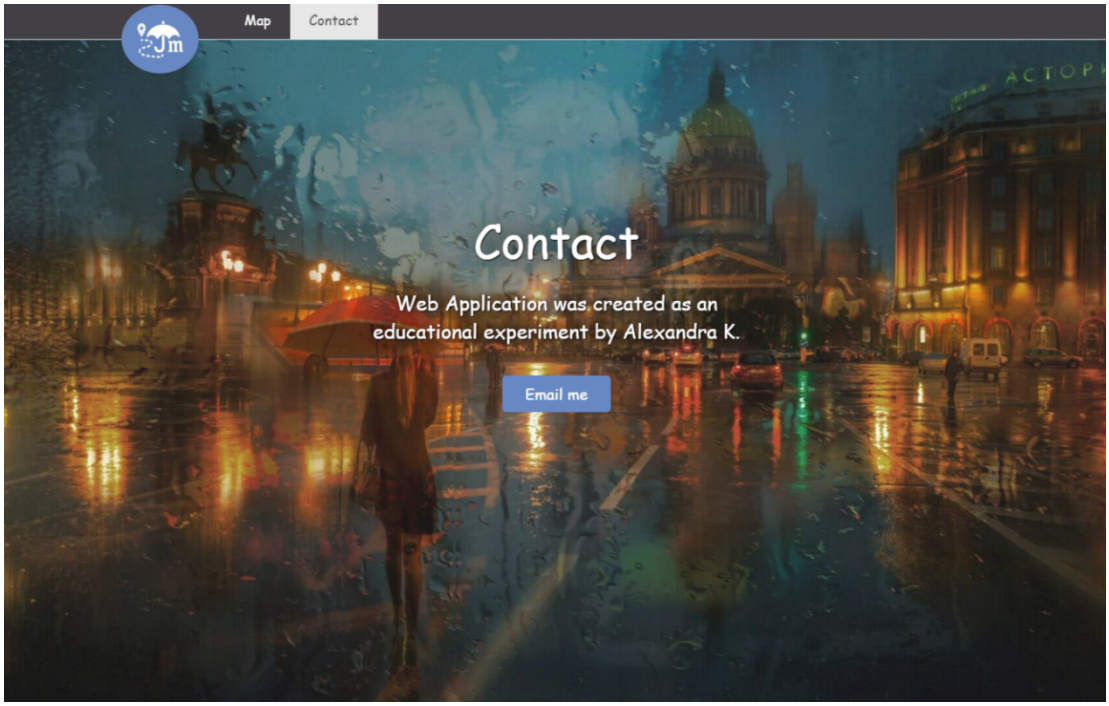

Рисунок 3.6. Третья секция приложения JourneyMap с обратной формой.

Ниже представлен краткий алгоритм работы JavaScript кода для непосредственного поиска оптимального маршрута на карте, представленной во второй секции:

1. инициализируем Google.Maps с определенными параметрами;
2. в результате взаимодействия считываем координаты точек и инициализируем начальную популяцию из заданных узлов;
3. оцениваем каждую особь данной популяции с учетом приспособленности хромосом (длины маршрута);
4. выбираем особей с наилучшей хромосомой;
5. скрещиваем выбранные особи;
6. мутируем полученное поколение, добавляя случайность;
7. повторяем предыдущие шаги, пока количество поколений не превысит 50;
8. готовый маршрут (особь с наилучшей хромосомой) добавляем на карту.

3.3. Анализ полученных результатов

Для повышения эффективной работы веб-сервиса провелось исследование, целью которого было найти параметры генетического алгоритма, гарантирующие наименьшую длину маршрута при минимальном затраченном на поиск маршрута времени в секундах. Приложение было протестировано на 2,2 GHz Intel Core i7 MacBook при различных значениях численности популяции, частоты мутации и количестве узлов меньшем 10. Таблица 3.1 показывает избранные результаты тестирования при различных параметрах.

Результаты тестирования JourneyMap при различных параметрах				
Количество узлов	**Численность популяции**	**Частота мутации**	**Время прохождения маршрута в минутах**	**Расчетное время в миллисекундах**
5	10	0.01	246.82	525936.695
5	100	0.01	240.60	594801.010
5	10	0.1	240.60	190642.445
5	50	0.1	240.60	280735.970
5	100	0.1	240.60	320717.775
5	10	0.5	240.60	418961.630
5	50	0.5	240.60	455337.725
5	100	0.5	240.60	472333.250
9	10	0.01	347.33	979128.000
9	50	0.01	347.80	1019110.910
9	100	0.01	345.37	1040919.235
9	10	0.1	328.97	1069626.310
9	50	0.1	319.52	1105175.810
9	100	0.1	320.48	1133307.290
9	10	0.5	355.80	1160534.845
9	50	0.5	350.52	1209408.045

Таблица 3.1 Результаты тестирования JourneyMap при различных параметрах: количестве узлов, численности популяции и частоте мутации.

Приведенные выше данные показывают, что при небольшом количестве узлов численность популяции и частота мутации в меньшей мере влияют на непосредственно длину маршрута, но заметно увеличивают расчетное время.

При количестве узлов близком к 10 наиболее близкие к желаемым значения получены при частоте мутации и численности популяции 0.1 и 50 соответственно. Суммируя вышесказанные – при работе приложения

JourneyMap оптимальные показатели были достигнуты при значениях численности популяции равной 10 и частоты мутации 0.5.

JourneyMap также было протестировано на потенциальных пользователях – результаты показали, что опыт взаимодействия (UX) был оценен как положительный.

Популярный среди туристов маршрут “Петропавловская крепость – Стрелка Васильвеского острова – Дворцовая площадь – Исаакиевский Собор – Казанский собор – Летний сад – Марсово поле – Петропавловская крепость” занимает 103 минуты, построенный с использованием Google.Maps. Маршрут, состоящий из посещения указанных выше мест, построенный в JourneyMap, может быть пройдет за 47 минут. Сравнение с Meganavigator, Логист и Speedy Route не может быть произведено ввиду отсутствия у них пешего режима.

3.4. Выводы по Главе 3

В главе 3 были перечислены инструменты, с помощью которых создавалось приложение (HTML5, CSS3, Bootstrap, JavaScript, jQueary, Goggle Maps и WebGL) и обосновано их присутствие. Приводится описание взаимодействия с JourneyMap со стороны пользователя. Дана структура JavaScript кода, строящего оптимальный экскурсионный маршрут, и рассказано о проделанной работе по повышению его эффективности, заключающейся в выборе оптимальных параметров генетического алгоритма.

ЗАКЛЮЧЕНИЕ

В данной работе была исследована задача коммивояжера. Были выявлены недостатки существующих приложений для поиска оптимального пути и по результатам исследования сформулированы требования к разрабатываемому приложению.

При решении задачи поиска оптимального пути были изучены различные алгоритмы, был проведен анализ, в результате которого был выбран генетический алгоритм как наиболее удовлетворяющий поставленные цели.

Генетический алгоритм был реализован на языке JavaScript с использованием Google.Maps и WebGL. Разработанное приложение было успешно протестировано при различных вводных данных, было исследовано качество результатов при определенных параметрах алгоритма, в результате чего были выбраны наиболее эффективные решения.

Цель работы, заключающаяся в создании удобного интерактивного приложения по поиску оптимального пути для экскурсоводов и туристов, была в полном объеме достигнута.

СПИСОК ЛИТЕРАТУРЫ

1. Geunes J. Operations Planning: Mixed Integer Optimization Models (Operations Research Series). CRC Press, 2014. P. 167–172.
2. Яндекс.Карты. https://yandex.ru/maps
3. Google Maps https://maps.google.com/
4. WebGL public wiki https://www.khronos.org/webgl/wiki/Main_Page
5. Cook W. J. In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton University Press, 2012. P. 19–39.
6. Applegate D.L., Bixby R.E., Chvátal V., Cook W.J. The Traveling Salesman Problem. Princeton University Press, 2007. P. 44–52.
7. Левитин А. Алгоритмы. Введение в разработку и анализ. Вильямс, 2006. P. 160.
8. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. Мир, 1982.
9. Okano H. Study of Practical Solutions for Combinatorial Optimization Problems. Tohoku University, 2009. P. 14–17.
10. Meganavigator. http://meganavigator.com/
11. Логист. http://logist.poncy.ru/
12. Speedy Route. https://www.speedyroute.com/
13. Левитин А. Алгоритмы. Введение в разработку и анализ. Вильямс, 2006. 35–36 с.
14. Kona H., Burde A., Dr. Zanwar D. R. A Review of Traveling Salesman Problem with Time Window Constraint // IJIRST – International Journal for Innovative Research in Science & Technology, 2015. Vol. 2, Issue 1. P. 253–254.
15. Tannenbaum P. Excursions in Mathematics. University of Kansas, 2011. P. 25.
16. Clausen J. Branch and Bound Algorithms – Principles and Examples. University of Copenhagen, 1999. P. 5-6.

17. Tollis I. G. Algorithms and Complexity. University of Crete, 2000. P. 140–146.
18. Applegate D., Bixby R., Chvatal V., Cook W. TSP cuts outside the template paradigm. Donet, 2000. P. 1–10.
19. Захарова Е.М., Минашина И.К. Обзор методов многомерной оптимизации // Информационные процессы, 2014. Том 14, № 3. стр. 265–266.
20. Papadimitriou C., Vazirani U. Efficient Algorithms and Intractable Problems. Berkeley EECS, 1999. Chapter 10.
21. Goodrich M., Roberto T. Algorithm Design and Applications, Wiley, 2015. P. 513–514.
22. Nilsson C. Heuristics for the Traveling Salesman Problem. Linkoping University, 2011. P. 1–6.
23. Alsalibi B.A., Jelodar M.B., Venkat I. A Comparative Study between the Nearest Neighbor and Genetic Algorithms: A revisit to the Traveling Salesman Problem // International Journal of Computer Science and Electronics Engineering (IJCSEE), 2013. Vol. 1, Issue 1.
24. Gutin G.,Yeo A. The Greedy Algorithm for the Symmetric TSP. University of London, 2002. P. 1–2.
25. Gutin G., Karapetyan D. Lin-Kernighan Heuristic Adaptations for the Generalized Traveling Salesman Problem. Royal Holloway London University, 2010. P. 1–5.
26. Gupta R., Chauhan C., Pathak K. Survey of Methods of Solving TSP along with its Implementation using Dynamic Programming Approach. // International Journal of Computer Applications, 2012. Vol. 52, No.4. P. 1–6.
27. Basu S. Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey. Indian Institute of Management Calcutta, 2012. P. 1–8.
28. Dorigo M., Caro G. D. Ant Algorithms for Discrete Optimization // Artificial Life, 1999. Vol. 5, No 2. P. 139–140.

29. Dorigo M., Gambardella L. M. Ant colonies for the traveling salesman problem. Université Libre de Bruxelles, 1996. P. 1–4.
30. Johnson D. S., McGeoch L. A., Rothberg E. E. Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound. AT&T Bell Laboratories, 1999. P. 1–2.
31. Abdulkarim H.A., Alshammari I. F. Comparison of Algorithms for Solving Traveling Salesman Problem. // International Journal of Engineering and Advanced Technology, 2015. Vol.4, Issue 6. P. 76–79.
32. Роббинс Д.Н. HTML5, 5-е издание. Вильямс, 2015. P. 11–14.
33. HTML5. W3C Recommendation. https://www.w3.org/TR/html5/
34. Макфарланд Д.С. Большая книга CSS3. Символ-Плюс, 2014. P. 10–24.
35. Stylus official site http://stylus-lang.com/
36. Bootstrap official site http://getbootstrap.com/
37. Флэнаган Д. Javascript. Подробное руководство. 6 издание. Символ-Плюс, 2013. P. 21–39.
38. jQuery official site https://jquery.com/
39. Google Maps API https://developers.google.com
40. Мацуда К., Ли Р. WebGL: Программирование трехмерной графики. ДМК Пресс, 2015. P. 26-41.